

Frequently Asked Questions (FAQs)

for

Murach's Structured COBOL

Where do I get support for Micro Focus Personal COBOL if I'm having a problem with it?

You can get support for Personal COBOL from Micro Focus by sending an e-mail to pcobsup@microfocus.com. The Micro Focus web site is found at www.microfocus.com.

Although we've run Personal COBOL on several operating systems without any problems, there is a problem with Windows 2000 and with Windows XP that can be fixed by downloading a file as described in the next FAQ.

Incidentally, Micro Focus doesn't plan to come out with a new release of Personal COBOL, which was developed for Windows 3.1. Instead, Micro Focus is replacing Personal COBOL with the University Edition of their Net Express product. This is an excellent product that has an Animator that works just like the one for Personal COBOL (the one that's described in chapter 2 of our book) so you can easily use it in place of Personal COBOL.

For college courses, we package our book with either Personal COBOL or the University Edition of Net Express.

How do I get Micro Focus Personal COBOL to run under Windows 2000 or Windows XP?

1. Go to this web page at Micro Focus:
www.microfocus.com/academic/pcw003.asp?bhcp=1
2. Click on the download link entitled ANIM2WG.EXE (64 k).
3. Once the file is downloaded, move it to the PCOBWIN folder on your hard drive, replacing the old file with the new file.
4. Reboot the PC.

Note that Micro Focus mentions that the object-oriented COBOL features of Personal COBOL may fail under Windows 2000 or XP. So be aware that you may run into problems when you use those features.

Why should I get Micro Focus Personal COBOL if I'm learning COBOL for an IBM mainframe?

First, we recommend the use of Micro Focus Personal COBOL because it's the cheapest and best product for developing and running COBOL programs on your own PC. Since COBOL is a standard language, it's essentially the same whether you run it on a PC or a mainframe. And Personal COBOL provides an efficient way to learn it.

Second, the Personal COBOL Animator that's described in chapter 2 of our book lets you step through programs so you can see what happens as each statement is executed. That's a great way to learn, and you can't duplicate that on a mainframe.

Once you're comfortable with the COBOL language, you can use our book to learn all of the related skills that you need for running COBOL on a mainframe. For instance, chapter 18 shows you how to compile and test a COBOL program on a mainframe...far more complicated than doing that with Personal COBOL. And chapters 19 and 20 introduce the skills for developing interactive and database programs for a mainframe. That's where the real complexity comes in.

Of course, if you have access to a mainframe, you don't need to use Personal COBOL. You can just read chapter 18 after chapter 2 and enter and run your programs on a mainframe. That, however, is a cumbersome way to get started. We think you'll find it's much more efficient to learn COBOL using Personal COBOL.

How do you pass a parameter from OS/390 JCL on a mainframe to a COBOL program?

In the Linkage Section of the COBOL program, you code the receiving field for the parameter or parameters that you want passed from the JCL as in this example:

```
LINKAGE SECTION.  
01  PARM-REC.  
    05  PARM-LENGTH          PIC S9(04)  COMP.  
    05  PARM-CLASS           PIC X(10) .  
    05  PARM-YEAR            PIC X(04) .
```

Here, PARM-CLASS and PARM-YEAR are the receiving fields for the parameters that are passed. But note that you also have to code a length field defined as S9(04) with COMP usage that receives the length of the parameter or parameters that are passed. If you don't code the length field, the first two bytes of the field will be treated as the length, which in effect truncates those bytes from the parameters.

Then, in the Procedure Division, you can use the passed fields just as you use any other fields. If, for example, you want to pass the parameters to the heading line of a report, you can use MOVE statements like this:

```
MOVE PARM-CLASS TO H1-CLASS.  
MOVE PARM-YEAR  TO H1-YEAR.
```

Later, when you code the JCL that runs the program, you use the PARM parameter in the EXEC statement for the COBOL program to pass the parameter or parameters as shown in this example:

```
//STULIST EXEC PGM=STUDLIST,PARM='SENIORS 2002'
```

Here, SENIORS and 2002 are passed to the COBOL report program named STUDLIST so they can be used in the heading of the report. If you have any difficulty with this JCL or if you want to enhance your JCL skills, we recommend our new JCL book called *Murach's OS/390 and z/OS JCL*, which should be available in March 2002.

Why doesn't your book present the balance-line algorithm for sequential file maintenance programs?

In case you aren't familiar with the balance-line algorithm, it is one of the many solutions that has been proposed for sequential file maintenance programs that need to process additions, deletions, and changes. If you've ever written one of these programs, you know that the logic gets nasty and there really isn't an elegant solution to it.

In our book, we present Paul Noll's solution for this problem, one that he developed in the early 1970s when he was working for Pacific Telephone. Although we considered presenting the balance-line algorithm too, we ruled it out for two reasons. First, I think Paul's solution is easier to understand. Second, Paul's solution is widely used in industry because he promoted it as a structured programming consultant for more than 20 years.

Having said that, there's nothing wrong with using the balance-line algorithm. What's important is that you find or develop a solution that you understand, and then test it to make sure it works under all combinations of changes, deletions, and additions. Once you've done that, you should use that solution in every sequential maintenance program that you write.

Why don't you use "prime reads" for summary reports and sequential update programs the way other books do?

In case you aren't familiar with "prime reads," here's the way one of our customers presented this question:

"When I learned COBOL in college, we were taught about the 'priming read,' where the initialization paragraph calls the read paragraph, and then the main looping mechanism follows a 'process-read-process-read' pattern until end of file. The main advantage is that you don't need to check for 'At End' after every read because the Perform Until EOF does that for you.

"When I came to my new company, though, nobody ever heard of 'priming reads.' I believe it is because the Murach books have become the standard for many mainframe shops, such as mine. Do you know of this convention, and is there a particular reason why you don't use priming reads?"

My answer is that one of the principles of structured programming that we adhere to is this: Each module (or COBOL paragraph) should do one and only one function. That's why we don't separate the critical logic of a program by putting the first read for a file in one paragraph (the prime read) and all of the other reads in another paragraph. That's dividing a function between two paragraphs.

From a practical point of view, of course, it's not bad to use a prime read in a simple program because it does simplify the code a bit. But as a program gets more complex, you want each paragraph to represent an independent module so you can read it without having to refer to other paragraphs for additional information.

Incidentally, using prime reads is just one of the ways the programs in other books violate the principle of module independence. Those programs often violate the principle of proper subordination too. In fact, we think that some of those programs completely misrepresent the way programs are developed in the real world. (Although you can find ugly programs in the real world too, you wouldn't expect to find them in textbooks).